THE EFFECT OF BUFFER SIZE ON PAGES ACCESSED IN RANDOM FILES

By: Prashant C. Palvia

Palvia, P. "The Effect of Buffer Size on Pages Accessed in Random Files," Information Systems, Vol. 13, No. 2, 1988, pp. 187-191.

Abstract:
Prior works, for estimating the number of pages (blocks) accessed from secondary memory to retrieve a certain number of records for a query, have ignored the effect of main memory buffer size. While this may not cause any adverse impact for special cases, in most cases the impact of buffer sizes will be to increase the number of page accesses. This paper describes the reasons for the impact due to a limited buffer size and develops new expressions for the number of pages accessed. The accuracy of the expressions is evaluated by simulation modeling; and the effects of limited buffer size are discussed. Analytical works in database analysis and design should use the new expressions: especially when the effect of the buffer size is significant.

Article:

I. INTRODUCTION

Many file organizations and access methods are currently in use and have also been discussed in the database literature [1-5]. The applicability of a particular file organization and/or access method is mainly determined by the characteristics of user data requirements. Tools have been developed for automating and aiding the process of selecting the above database design parameters [6-9]. In many of these tools, an important task is the direct estimation of the number of pages (blocks) needed to satisfy a given user query. The number of pages is a simple function of the number of records needed, if each record is individually retrieved. However, if all records to be retrieved can be batched, then researchers have shown significant savings in sequential files [1, 10, 11], in hierarchical files [10, 11] and random files [12- 16].

In all of the above works, the effect of the size of the main memory buffer (which will hold the pages retrieved from secondary memory) has been ignored. In sequential and hierarchical files, this does not pose a problem and the expressions of [10, 11] are valid. This is because, to take advantage of batching, the required records are to be in sequence order; consequently once a page has been brought into main memory buffer and later released, it is not needed again.

The expressions of [12, 14-16] really give the average number of pages that will contain the required *k* random records in a random file organization. The expressions then imply that the same number of pages will have to be accessed into main memory. This naturally makes the assumption that once a page is accessed into the buffer, it will stay there until all required records from that page have been used by the calling program. This can happen only in the following cases:

> (i) The records are retrieved in their physical order. This is possible if the physical order is known (e.g. in absolute or relative addressing when primary key = absolute/relative address). However, as most modern database and file organizations are random (based on hashing the primary key), the user/program would not know the physical address; only the internal file system and access method will determine that. While there may be certain ways of arranging the required record keys in physical order, the most likely scenario is of presenting a batch of keys in no particular (random) order.

(ii) The buffer size is very large. If the record keys are presented in a random order, then the only way to ensure that a page, which has been brought into the buffer once and is needed later, exists in the buffer is to have very large buffer space. Even though main memories are getting cheaper, this is generally not possible in all cases. With smaller buffers, previously accessed pages will have to be "paged out" to make room for new pages.

Given that both of the above conditions are not feasible at all times, there will be additional "paging" activity to retrieve some pages which have been previously retrieved and later released. Thus the pages accessed to retrieve *k* records may be more than reported in the expressions of [12, 14-16]. En this paper, we develop expressions for estimating the average number of pages accessed into the main memory buffer (of finite size S) for retrieving *k* records from a randomly organized file. The results of the paper thus have consequences for many analytical works in databases, both prior and future.

The rest of the paper is organized as follows: in Section 2, we develop the appropriate expressions. In Section 3, we present data on page accesses for given file parameters and buffer sizes, discuss the results and compare them with results of earlier works. In Section 4, the expressions of Section 2 are evaluated for their correctness and accuracy using simulation modeling. The simulations also provide important data on the variability of the results. Finally, Section 5 concludes the paper.

2. THE EXPRESSIONS

Let the random file contain *n* records organized in *m* pages (blocks) of secondary memory, so that the blocking factor $p = n/m$. Further let *l* be the record length in characters and *S* be the buffer size in characters.

Then, buffer size in pages $B = S / pl$.

Further assume that the k records satisfying the query are distributed uniformly over the *n* records (as has been the assumption in prior works). Then the average number of distinct pages, *P*, containing the *k* records are given by the expressions in [12, 14-16]. We use the expression of [14] as it was shown to be both simple and quite accurate. According to [14],

$$P = n/p\ [1 — (1 – k/n)^p] \qquad (1)$$

If *P* is less than *B*, the buffer capacity, then all required pages can coexist in the buffer at the same time and there would be no further paging, (Actually there might be same additional paging on the average, because *P* represents the average case. Even when the average *P* is less than *B*, in some instances it may be greater than *B* thus causing additional paging. We do not consider this case because of its complexity and only marginal improvements).

Thus, if $P \leq B$, total pages accessed into the buffer is:

$$T = P. \qquad (2)$$

We now take the case when *P* is greater than *B*. The expression (1) gives the number of pages *P* on which *k* random records reside, The same expression can be inverted to determine the number of random records k_1 that reside on P_1 pages. Rewriting the expression (1), after substituting k_1, for *k* and P_1 for *P*, we have:

$$k_1 = n\ [1 — (1 — P_1 p/n)^{1/p}].$$

Using the above equation, if the buffer was full with *B* pages of the required file, it will contain *Q* required random records, where *Q* is given by:

$$Q = n\ (1 — (1— Bp/n)^{1/p}). \qquad (3)$$

Since *B* is less than *P, Q* is less than *k.* In other words, *Q* represents the average number of records present in the buffer out of the total *k* records required by the query, when the buffer is filled for the first time. The remaining k - *Q* records required by the query may or may not be present in the buffer and may require further page accesses. However, any new page accesses will require that the buffer give up some of its current pages. So we now need to estimate how many additional pages (in addition to the *B* pages already accessed) need to be accessed to get the balance of *k* - *Q* records.

Each of the *k* - *Q* records can be inside or outside of the buffer. So we have to estimate the probability of it being inside the buffer. The buffer has *B* pages, each containing *p* records, or a total of *Bp* records. Initially, on the average, of these *Q* records have already been selected. Since, the *k* records to be retrieved are distinct, only the remaining *Bp* — *Q* records qualify for selection from the buffer. Also since *Q* records have already been selected, the remaining desired records would have to come from the remaining total of n - *Q* records.
Thus, the probability of the buffer containing an additional desired record:

$$= (Bp - Q)/(n - Q). \qquad (4)$$

The probability of the buffer not containing the desired record is one minus the above probability, which is = (*n* - *Bp*)/(*n* - *Q*). This is the probability with which a new page would have to be accessed for each additional desired record. Since there are *k* - *Q* more records to retrieve, additional pages accessed *A*1 is:

$$A1 = (n - Bp)\cdot(k - Q)/(n - Q). \qquad (5)$$

Actually, we used a simplifying argument in the equation (4). We assumed that the total remaining records will be *n* - *Q* for ee.ch additional desired record. This is true only for the first desired record after the buffer is full for the first time. For each additional desired record after that, the total remaining would be one less. If we are searching the *i*th desired record after the buffer was full for the first time, then the total remaining records are *n* - *Q* - (*i* - 1) and the probability of the buffer containing the desired record is:

$$= (Bp - Q)/[n - Q - (i - 1)]. \qquad (6)$$

Using the same arguments as before, additional pages accessed:

$$A2 = \sum_{i=0}^{j} (n - Bp - i)/(n - Q - i),$$

$$\text{where } j = k - Q + 1. \qquad (7)$$

Note that *j* cannot be greater than *n* — *Bp* (because that results in accessing more records than are in the entire file). Also note that when *Bp* is small compared to *n,* and *j* (or *k)* is small compared to *n,* then the equation (7) can be approximated by equation (5). Both of these assumptions will hold in most queries. In any case the following is a good approximation of (7), in that it takes an "average" of the *k* — *Q* terms.

$$A2 = (k - Q).\frac{[n - Bp - (k - Q)/2]}{[n - Q - (k - Q)/2]}. \qquad (8)$$

The approximation of equation (6) can be further improved. We said that any time there are an average *Q* records on the buffer, which have already been selected. Actually this is true only when the buffer was first filled. As more and more records are being selected, the previously-selected records on the buffer will also increase. When all records for the query have been selected, then the buffer would have *Bk / P* preselected records (because the *P* pages contain all *k* records and the buffer has *B* pages). Thus the preselected records on the buffer will range from *Q to Bk / P.* Therefore a better measure of preselected records on the buffer at any given time is the average of *Q* and *Bk / P,*

$$\text{or } Q1 = (Q + Bk/P)/2, \qquad (9)$$

With this the number of additional accesses $A3$ is recomputed using previous arguments and approximations, so that we have:

$$A3 = (k - Q).\frac{[n - Bp - (k - Q)/2 + Q1 - Q]}{[n - Q - (k - Q)/2]}. \quad (10)$$

Given values for $A1$, $A2$ and $A3$, we have three approximations for the number of page accesses to retrieve k records in a buffered environment, when $P > B$, then:

$T = B + A1$, $A1$ defined by equation (5) (11)

$T = B + A2$, $A2$ defined by equation (8) (12)

$T = B + A3$, $A3$ defined by equation (10) (13)

and when $P \leq B$, then:

$T = P$ (from equation 2),

We now use the above equations to compute accesses for sample file data., discuss results and compare the results with results of prior works.

DISCUSSION

Table 1 shows the number of pages accessed for a file with 300 records and record length of 100 characters, as computed using the expressions of this paper. Column A shows the number of accesses when searching each record individually (i.e. without batching, when number of pages simply equal the number of records required); columns B, C, D and E show the page accesses with buffer sizes of 1000. 2000, 4000 and 10,000 characters; and column F shows the page accesses with unlimited buffer size. The last column F represents the expressions already in the literature [12, 14-16] which do not consider buffers at all. We used equation (1) in obtaining the last column. Average page accesses in columns B, C, D and E were obtained using equation (13) (combined with equation 2). Although we do not report the results of equations (11) and (12), they were less than 1% off from the equation (13) results in all except one of the 60 reported cases. Thus any of the equations (11), (12) or (13) may be used; perhaps equation (11) should be preferred because of its relative simplicity.

For each column, results are reported for different blocking factors (p = 1, 5, 10) and different number of records accessed (k = 2, 5, 10, 20, 50). Based on this data, we make the following observations. Many of these observations formalize current knowledge in the area.

A.
Column A (individual searches) shows the most number of pa.ge accesses and column F (infinite buffer) shows the least number of pages accesses. However in many online fast-response queries, one does not have the option of hatching queries and only the "individual search" option is available. On the other hand, the luxury of having extremely large buffers is not always available; thus the number of page accesses are generally larger than indicated by column F and as reported in [12, 14-16].

B.
The number of page accesses increases with the reduction in buffer size. When the buffer size is very small, the number of page accesses are very near the number of page accesses with individual searches and no advantage is gained by batching. With increasing buffer size, the number of page accesses approach the case of infinite buffer size; and more and more advantage of batching is obtained.

C.
As has been observed in the case of sequential and hierarchical files [10, 11], the advantages of batching get more significant for higher number of records to retrieve. This is intuitive as batching will naturally pay off more when there is more to batch.

Table 1. Expected pages accessed in random files (as computed from the analytical expressions)

k	p	A Individual searches	B Buffer = 1000 char	C Buffer = 2000 char	D Buffer = 4000 char	E Buffer = 10,000 char	F Infinite buffer
$k = 2$	$p = 1$	2	2	2	2 2	2	
	$p = 5$	2	1.97	1.97	1.97	1.97	1.97
	$p = 10$	2	1.96	1.94	1.94	1.94	1.94
$k = 5$	$p = 1$	5	5	5	5	5	5
	$p = 5$	5	4.89	4.84	4.84	4.84	4.84
	$p = 10$	5	4.86	4.76	4.65	4.64	4.64
$k = 10$	$p = 1$	10	10	10	10	10	10
	$p = 5$	10	9.76	9.57	9.37	9.36	9.36
	$p = 10$	10	9.71	9.46	9.04	8.63	8.63
$k = 20$	$p = 1$	20	20	20	20	20	20
	$p = 5$	20	19.49	19.02	18.28	17.51	17.51
	$p = 10$	20	19.41	18.84	17.81	15.59	14.95
$k = 50$	$p = 1$	50	50	50	50	50	50
	$p = 5$	50	48.64	47.34	44.9	39.11	35.89
	$p = 10$	50	48.47	46.97	44.06	36.18	25.16

†k = number of records required; p = blocking factor = n/m; s = buffer in size characters; l = record size = 100 characters; n = number of records = 300.

D.
Another observation is that batching will pay off only when there is blocking of the records in secondary memory. With no blocking ($p = 1$), all columns show the same number of page accesses (i.e. equal to that in the case of individual searches). This is because, with distinct records to search and one record per page, a new record or page would have to be retrieved for a new record. With higher blocking factors, there will be a possibility of a block having other desired records, other than the record for which the page was originally retrieved. Thus with higher blocking factors, the number of pages accessed decreases.

We now evaluate the validity of the expressions developed in Section 2.

4. EVALUATION OF RESULTS

Since, in order to obtain the expressions of Section 2, many approximations were made, the integrity of the expressions may be in question. The same goes for the approximations [12, 14] in the unbuffered case; but mathematically proven exact expressions [15, 16] are available in that ease and the accuracy of the approximations has been demonstrated. No mathematically provable expressions exist in the limited buffer case and are probably very hard to develop. Therefore a simulation program was written in BASIC on a personal computer to compute the number of page accesses in a buffered environment. The simulation results are shown in Table 2.

The simulation provides data not only on the averages but also on the variability (the variability data has not been reported earlier in the literature). Both the arithmetic average and standard deviation are reported for buffer sizes of 1000, 2000, 4000 and 10,000 characters, The data reveals amazing closeness between the simulation results and analytical results. The results are so close that no statistical tests were deemed necessary. In fact the maximum difference in the two sets of results is 2% with the average absolute difference being about 0.5%. It was also observed that the differences were further reduced as the number of simulation nuts per data point was increased. Also there is no readily identifiable pattern in the differences; they are sometimes positive and sometimes negative. We contend, therefore, that the expressions developed in Section 2 of this paper are quite accurate.

Some general observations are made on the pattern of the standard deviations. First, all standard deviations are relatively low indicating low variability in the number of pages accessed. This is good news for analytical works which are based on averages alone. Such works have been criticized as they fail to incorporate the effects

of variability; but if the variability is low, then such effects would be minimal. Second, perhaps contrary to intuition, the variability is higher in the number of pages accessed for higher buffer sizes than lower buffer sizes, This is due to the reason that for small buffer sizes, the number of pages accessed is so close to the maximum that there is little room for much variability. Third, while the standard deviation increases with increasing average number of pages, the coefficient of variation (standard deviation divided by the average) remains same or goes down, This further corroborates the relatively minor impact of variability in analytical studies.

Table 2. Simulation results for expected pages accessed in random files

		A buffer = 1000 char		B buffer = 2000 char		C buffer = 4000 char		D buffer = 10,000 char	
k	p	Avg	SD	Avg	SD	Avg	SD	Avg	SD
$k = 2$	$p = 1$	2	0	2	0	2	0	2	0
	$p = 5$	2	0	2	0	2	0	2	0
	$p = 10$	2	0	2	0	2	0	2	0
$k = 5$	$p = 1$	5	0	5	0	5	0	5	0
	$p = 5$	5	0	5	0	4.8	0.40	4.8	0.40
	$p = 10$	4.9	0.3	4.8	0.37	4.7	0.44	4.5	0.67
$k = 10$	$p = 1$	10	0	10	0	10	0	10	0
	$p = 5$	9.8	0.43	9.5	0.50	9.5	0.61	9.5	0.67
	$p = 10$	9.7	0.30	9.6	0.49	9.4	0.80	9.0	1.00
$k = 20$	$p = 1$	20	0	20	0	20	0	20	0
	$p = 5$	19.5	0.56	19.2	0.75	18.8	0	20	0
	$p = 10$	19.2	0.68	19.3	0.78	17.6	1.2	15.5	1.62
$k = 50$	$p = 1$	50	0	50	0	50	0	50	0
	$p = 5$	48.5	0.92	47.1	1.14	45.0	1.9	39.7	2.05
	$p = 10$	48.5	0.92	46.7	1.19	44.6	1.8	35.3	2.41

†k = number of records required; p = blocking factor = n/m; s = buffer in size characters; l = record size = 100 characters; n = number of records = 300; Avg = arithmetic average; SD = standard deviation.

Lastly, it should be pointed out that the simulation results (or results in actual operations) will depend on the "page replacement" strategy in the buffer space. This refers to the policy on which page to give up when a new page is required in the buffer. Two policies were implemented for the simulation: first in first out (FIFO) and last in first out (LIFO). The LIFO policy resulted in much higher number of pages than the FIFO policy. Only the FIFO results are reported in Table 2. It does make sense to use the FIFO policy, as it is fair to all pages and also largely reflects actual practice.

5. CONCLUSIONS

This paper has reported on expressions developed for the number of pages accessed for retrieving a batch of records from a random file, when buffer sizes are limited. Previous results have not considered buffer sizes at all and are only applicable when either the buffer sizes are very large or the records can be retrieved in physical order. The results show that hatching still yields savings in the number of page accesses, although not nearly as much as reported in the previously published expressions [12, 14-16] The results have important consequences for both prior and future analytical works in databases.

REFERENCES

[1] D. S. Batory and C. C. Gotlieb. A unifying model of physical databases. *ACM Trans. Database Systems* 7(4) (December, 1982).

[2] J. Hoffer. Database design practices for inverted files. *Inform. Mgmt.* 3 (1980).

[3] D. G. Severance. A parametric model of alternative file structures. *Inform. Systems* 1(2) (1975).

[4] D. G. Severance and J. V. Carlis. A practical approach to selecting record access paths. *ACM comput. Sum.* 9(4) (December, 1977).

[5] S. B. Yao. An attribute-based model for database access cost analysis. *ACM Trans. Database Systems* 2(1) (March, 1977).

[6] J. V. Carlis and S. T. March. A computer aided database design methodology. *Comput. Perform.* (December, 1983).

[7] J. Hoffer and A. Kovacevic. Optimal performance of inverted files. *Ops Res.*, Vol 30(2) (March- April* 1982).

[8] T. J. Gambino and R. Gerritsen. A data base design decision support system. *Proc. VLDB, ACM* (1977).

[9] S. T. March. Techniques for record structuring, *ACM Comput. Sum.* 3 (1983).
[10] P. Palvia. Expressions for batched searching of sequential and hierarchical files. *ACM Trans. Database Systems* 10(11) (March, 1985).
[11] B. Schneiderman and V. Goodman. Batched searching of sequential and tree structure files. *ACM Trans. Database Systems* 1(13) (September, i976).
[12] A. F. Cardenas. Analysis and performance of inverted database structures. *Commun. ACM18(5)* (May, 1975).
[13] T. Y. Cheung. Estimating block accesses and number of records in file management. *Commun. ACM* 25(7) (July, 1982).
[14] P. Palvia and S. T. March. Approximating block accesses in database organizations. *Inform. Process. Lett.* 19 (August, 1984).
[15] S. B. Yao. Approximating block accesses in database organizations. *Commun. ACM* 20(4) (April, 1977).
[16] P. C. Yue and C. K. Wong, Storage cost considerations in secondary index selection. *Int. J. Comput. inform. Sci.* 4(4) (1975).